MOBILE APPLICATION DEVELOPMENT - 2

UNIT – 1 : PROJECT STRUCTURE OF MOBILE APPLICATION

What is Dalvik Virtual Machine?

Firstly let us understand what a virtual machine is? It is basically a software implementation of a physical computer. This implementation works like a real physical computer. It even compiles and runs programs the same as a physical computer. It can be understood like an emulator. There are some issues with virtual machines too. One is that it is less efficient when compared to physical computers. Another issue is its performance, which is unstable when multiple virtual machines are working simultaneously on the same machine.

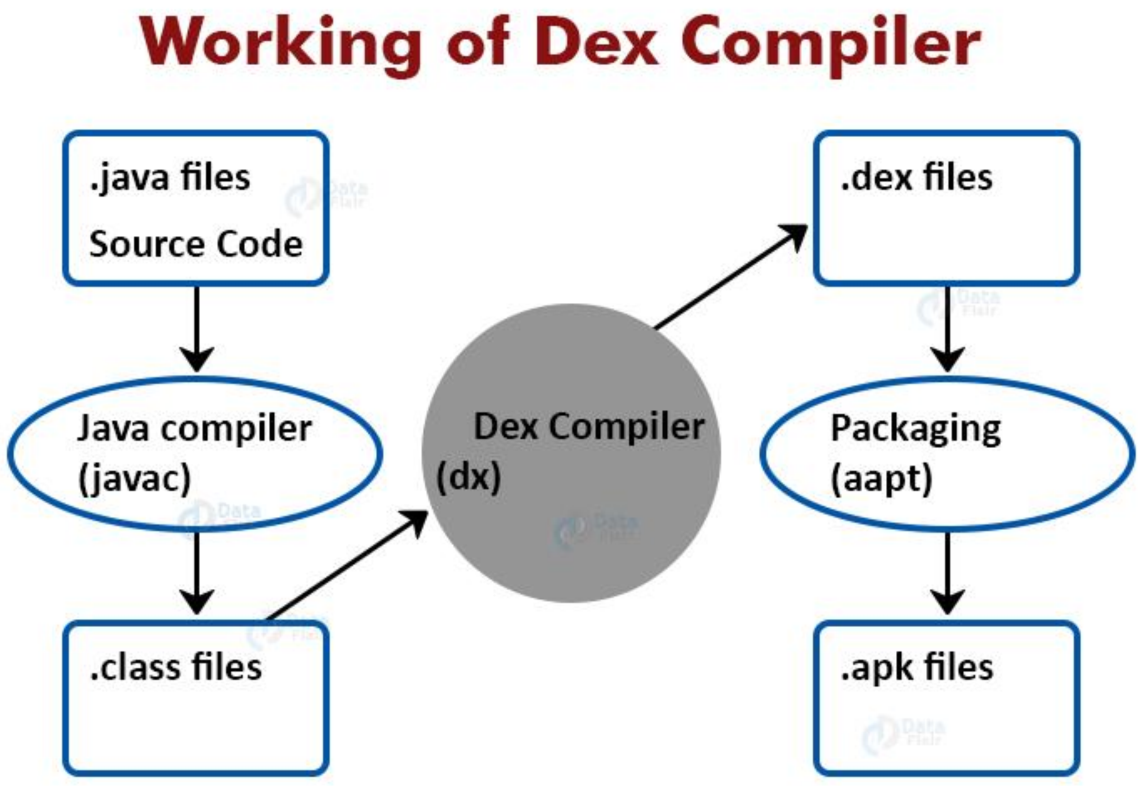
Even though Java Virtual machine has a high performance and provides great memory management, it is not optimized for low-powered devices. Dalvik VM is also a virtual machine that is highly optimized for mobile devices. Thus, it provides all the three things, that are **memory management, high performance as well as battery life**. It is strictly developed for Android mobile phones.
Role of the Dalvik Virtual Machine

The Role of the DVM in Android includes:

- Optimizing the Virtual Machine for memory, battery life, and performance
- Conversion of class files into .dex file through Dex compiler that runs on Dalvik VM.
- Converting multiple class files into dex files.

The Dex compiler helps convert the class file into .dex file, the following image shows how it flows:

- First of all the .java file converts into .class file with the help of Java compiler.
- Next .class file converts into .dex file using Dec compiler.
- Then finally the packaging process is handled by the Android Assets packaging (aapt) tools.



Dalvik Virtual Machine vs Android Runtime:

Let us see comparison between Android DVM and Android Runtime:

Dalvik Virtual Machine	Android Runtime
Dalvik is slower in comparison to Android Runtime	Android Runtime is faster than Dalvik Virtual Machine
Dalvik Virtual Mchine takes less time to boot, Booting is fast	Android Runtime takes more time to boot, Booting is slow
The Cache builds up fast over time, reducing the reboot time	The cache is built at first boot, increasing the reboot time
Dalvik Virtual Mchine needs less space as it uses Just In Time compiler	Android Runtime needs more space as it uses AOT.
Dalvik Virtual Mchine utilizes more battery, thus it has low battery performance	Android Runtime utilizes less battery, thus it has high battery performance
Dalvik Virtual Mchine has a poor Garbage collection when compared to Android Runtime	Android Runtime has a better Garbage collection when compared to DVM

In Android DVM, apps are less responsive in accordance with Android Runtime	Apps here are very responsive and work smoothly.
Dalvik Runtime Virtual Machine converts bytecode every time the application launches	On the other hand, Android Runtime converts the bytecode only once at the time of installation of application
It is a stable and time-tested virtual machine	It is highly experimented and new
DVM is the choice of Android developers	It doesn't have a lot of support from app developers until now
DVM works better for lower internal storage devices as space occupied is less	It consumes more internal storage space, as it stores compiled apps in addition to the APKs
It came prior to Android Runtime and it is replaced with Android Runtime	Android Runtime is the upgraded version of Dalvik Virtual Machine and comes up with a lot of improvements

2. Explain Screen Orientation in Android Studio.

The **screenOrientation** is the attribute of activity element. The orientation of android activity can be portrait, landscape, sensor, unspecified etc. You need to define it in the AndroidManifest.xml file.

Syntax:

```
<activity android:name="package_name.Your_ActivityName"
      android:screenOrientation="orirntation_type">
</activity>
```

Example:

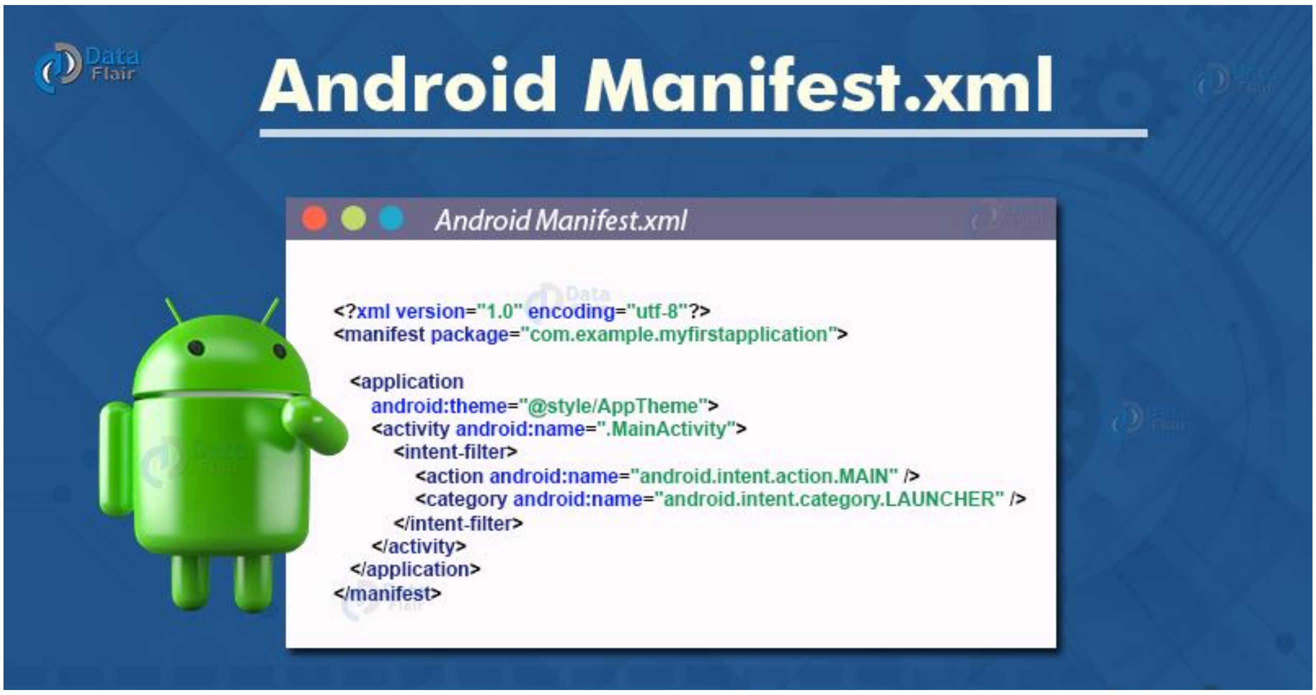
```
<activity android:name=" example.javatpoint.com.screenorientation.MainActivity"
    android:screenOrientation="portrait">
</activity>
<activity android:name=".SecondActivity"
    android:screenOrientation="landscape">
</activity>
```

The common values for screenOrientation attribute are as follows:

Value	Description
unspecified	It is the default value. In such case, system chooses the orientation.
Portrait	taller not wider
Landscape	wider not taller
Sensor	orientation is determined by the device orientation sensor.

3. What is Android Manifest File – androidmanifest.xml file?

As we know that every android project that we make, need to have an Android Manifest file within. This file is present in the root of the project source set. The reason why it is a must to have it is that it describes all the important information of the application to the Android build tools.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="com.example.myfirstapplication">

  <application
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Android Manifest File

The manifest file in Android is generally created automatically as soon as the app is built in Android Studio.

Structure of a Manifest file in Android is:

```
<manifest>
   <application>
```

```
<activity android:name="com.example.applicationname.MainActivity" >
       </activity>
   </application>
</manifest>
```

The information that is stored in the Manifest file is as follows:

- The name of the application's package, it is generally the code's namespace. This information is used to determine the location of the code while building the project.
- Another component is the one, that includes all the activities, services, receivers, and content providers.
- The permissions that are required by the application to access the protected parts of the system and other apps.
- The features required by the app, that affect which devices can install the app from Google Play. These features include both hardware and software features.
- It also specifies the application metadata, which includes the icon, version number, themes, etc.

The android manifest.xml file generally looks like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.myfirstapplication">

   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
       <activity android:name=".MainActivity">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
       <meta-data
           android:name="preloaded_fonts"
           android:resource="@array/preloaded_fonts" />
   </application>

</manifest>
```

Element Tags of Manifest File

Following are the essential element tags of Manifest.xml files:

1.<manifest>
It is the root element of this element. It consists of a package attribute package that tells the activity's package name.

2.<application>
It is the subelement of the manifest file that includes the declaration of the namespace. It contains certain attributes. These attributes declare the application components, and these attributes include:

- icon
- allowBackup
- label
- theme

3.<activity>
Activity is a subelement of application. It has the declaration of the activity that must be there in the manifest file. It also has certain attributes like name, label, theme, etc.

4.<intent-filter>
It is an element in the activity, it describes the type of intent in which the Android components can respond.

5.<action>
This element provides an action for the intent filter. Each intent filter must have at least one action element in it.

6.<category>
This element adds the category name in an intent-filter.

7.<service>
This element contains the operations that are provided by libraries or APIs.

Manifest File Application Property Elements

Following is a list of important Application Property Elements in manifest.xml (Sub-Node Elements)

1. <uses-permission>: This element specifies the Android Manifest permissions that are requested for the purpose of security.

2. <permission>: This element sets permission to provide access to control for some components of the app.

3. <permission-groups>: This element sets permission to provide access to control for a set of components of the app.

4. <permission-tree>: This element refers to a specific component that is the owner of the set of components.

5. <instrumentation>: This element tells the interaction between the app and the system.

6. <uses-sdk>: This one specifies the compatibility of the app.

7. <uses-configuration>: This element specifies the permissions that are requested for the purpose of security.

8. <uses-feature>: This element specifies one hardware or software feature that is required by the Application.

9. <supports-screen, compatible-screen>: These elements tell the screen size and configuration.

How to set Manifest File Permissions in Android?

Now in this part of the tutorial, we'll see how to set permission in a Manifest file.

An android application must get some permissions to get access to other apps or the Internet. While we build our app, the manifest file gets automatically generated as manifest.xml. This manifest file contains permissions that are configured by us. A few permissions are applied by default if there is no permission provided by us.

These are written in the manifest file as:

```
<manifest >
<uses-permission .../>
...
<manifest >
```

The permission mentioned in the manifest file can be either granted or rejected by the users. Once the user grants permission to the app, the app can use the protected features. If the user denies permission the app doesn't get permission to access the protected features.

The following permissions are added by default and set to TRUE:

- INTERNET
- ACCESS_NETWORK_STATE
- READ_PHONE_STATE

There are many permissions that are set to FALSE by default but can be set to TRUE as per requirements. A few of these permissions are as follows:

ACCESS_WIFI_STATE	AUTHENTICATE_ACCOUNT	BLUETOOTH	BATTERY_STATS
BIND_APPWIDGET	BROADCAST_WAP_PUSH	BROADCAST_STICKY	BIND_INPUT_METHOD
CALL_PHONE	CHANGE_CONFIGURATION	CAMERA	CLEAR_APP_DATA
CHANGE_WIFI_STATE	CLEAR_APP_USER_DATA	DEVICE_POWER	DELETE_CACHE_FILES
DISABLE_KEYGUARD	DELETE_PACKAGES	EXPAND_STATUS_BAR	EXPAND_STATUS_BAR
FACTORY_TEST	GET_PACKAGE_SIZE	FLASHLIGHT	GLOBAL_SEARCH
HARDWARE_TEST	INTERNAL_SYSTEM_PROCESSES	USE_CREDENTIALS	MANAGE_ACCOUNTS
MANAGE_APP_TOKENS	MODIFY_AUDIO_SETTINGS	MODIFY_PHONE_STATE	NFC
SEND_SMS	PROCESS_OUTGOING_CALLS	SET_ALARM	SET_ALWAYS_FINISH
READ_CALENDAR	KILL_BACKGROUND_PROCESSES	SET_WALLPAPER	VIBRATE
WAKE_LOCK	WRITE_APN_SETTINGS	WRITE_CALENDAR	WARITE_SETTINGS

4. What is R.java file in Android Studio?

Android R.java is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.

If you create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.

Let's see the android R.java file. It includes a lot of static nested classes such as menu, id, layout, attr, drawable, string etc.

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.example.helloandroid;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int menu_settings=0x7f070000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int activity_main=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello_world=0x7f040001;
        public static final int menu_settings=0x7f040002;
    }
    public static final class style {
        /**
        Base application theme, dependent on API level. This theme is replaced
        by AppBaseTheme from res/values-vXX/styles.xml on newer devices.


            Theme customizations available in newer API levels can go in
            res/values-vXX/styles.xml, while customizations related to
            backward-compatibility can go here.
        Base application theme for API 11+. This theme completely replaces
        AppBaseTheme from res/values/styles.xml on API 11+ devices.

    API 11 theme customizations can go here.

        Base application theme for API 14+. This theme completely replaces
        AppBaseTheme from BOTH res/values/styles.xml and
        res/values-v11/styles.xml on API 14+ devices.

    API 14 theme customizations can go here.
         */
        public static final int AppBaseTheme=0x7f050000;
        /**  Application theme.
    All customizations that are NOT specific to a particular API-level can go here.
         */
        public static final int AppTheme=0x7f050001;
      }
    }
```

5. What is Android Widgets?

A widget is a small gadget or control of your android application placed on the home screen. Widgets can be very handy as they allow you to put your favourite applications on your home screen in order to quickly access them. You have probably seen some common widgets, such as music widget, weather widget, clock widget e.t.c

Widgets could be of many types such as information widgets, collection widgets, control widgets and hybrid widgets. Android provides us a complete framework to develop our own widgets.

6. How to hide Title bar in Android App Development?

In this example, we are going to explain how to hide the title bar.

The requestWindowFeature(Window.FEATURE_NO_TITLE) method of Activity must be called to hide the title. But, it must be coded before the setContentView method.

Code that hides title bar of activity

The getSupportActionBar() method is used to retrieve the instance of ActionBar class. Calling the hide() method of ActionBar class hides the title bar.

- requestWindowFeature(Window.FEATURE_NO_TITLE);//will hide the title
- getSupportActionBar().hide(); //hide the title bar

Full Example that hides title bar

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.androi
d.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="first.javatpoint.com.hidetitlebar.MainActivity">
```

```
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE); //will hide the title
        getSupportActionBar().hide(); // hide the title bar
setContentView(R.layout.activity_main);

    }
}
```

7. Explain Screen Orientation in Android Studio.

The **screenOrientation** is the attribute of activity element. The orientation of android activity can be portrait, landscape, sensor, unspecified etc. You need to define it in the AndroidManifest.xml file.

Syntax:

```
<activity android:name="package_name.Your_ActivityName"
     android:screenOrientation="orirntation_type">
</activity>
```

Example:

```
<activity android:name=" example.javatpoint.com.screenorientation.MainActivity"
    android:screenOrientation="portrait">
</activity>
<activity android:name=".SecondActivity"
    android:screenOrientation="landscape">
</activity>
```

The common values for screenOrientation attribute are as follows:

Value	Description
unspecified	It is the default value. In such case, system chooses the orientation.
Portrait	taller not wider
Landscape	wider not taller
Sensor	orientation is determined by the device orientation sensor.

8. Explain Form Widget Palettes (Controls) in Android Studio.

The Android Studio palette contains various different views that we can drag onto the "design editor" representing the display of an Android device. These views are divided into categories and are all covered in the following sections.

1. TextView :

Android Textview is user interface element that shows texts to the user and optionally allow them to edit it. However, Text editing is disabled by default. We need to configure the basic class by adding some attributes to make it editable.

Different Attributes of Android TextView Widget

Below are the different attributes of android Textview widget that are commonly used to customise the Textview. However, you can check the android official documentation website for complete list of Textview attributes. Here, we are listing some of the attributes that we will need commonly.

Sr.	XML Attributes	Description
1	android:autoLink	Use this attribute when you want to automatically detect urls or emails and show it as clickable link.
2	android:autoText	If it is set, it means Textview has a textual input method and automatically corrects some common spelling errors.
3	android:capitalize	If it is set, it means Textview has a textual input method and should automatically capitalize what the user types.
4	android:id	This is unique id of the widget to uniquely identify the widget.
5	android:cursorVisible	This is used to make cursor visible or invisible. Default is false.
6	android:drawableBottom	Use this attribute when you want to show any drawable(images etc.) below the text.
7	android:drawableEnd	Use this attribute when you want to show any drawable(images etc.) to end of the text.
8	android:drawableLeft	Use this attribute when you want to show any drawable(images etc.) to left of the text.
9	android:drawablePadding	Use this attribute when you want to add padding to the drawable(images etc.).
10	android:drawableRight	Use this attribute when you want to show any drawable(images etc.) to right of the text.

11	android:drawableStart	Use this attribute when you want to show any drawable(images etc.) to start of the text.
12	android:drawableTop	Use this attribute when you want to show any drawable(images etc.) to top of the text.
13	android:ellipsize	This attribute causes the text, longer than the view, to be ellipsized at the end.
14	android:ems	Makes the Textview be exactly this many ems wide.
15	android:gravity	This is used to align the text (by x-axis, y-axis or both) within this Textview.
16	android:height	This is used to provide the height to the Textview.
17	android:hint	Hint to be shown when there is no text in the Textview.
18	android:inputType	This is used to define what are the types of data can be entered by the user for this Textview. For example, Phone, Password, Number, Date, Time etc.
19	android:lines	If you want to set height of the Textview by number of lines, you can do it using this attribute. For example, android:lines=”2”, it means height of Textview will be 2 lines.
20	android:maxHeight	Use to set the maximum height of the Textview.
21	android:minHeight	Use to set the minimum height of the Textview.
22	android:maxLength	Use to set the maximum character length of the Textview.
23	android:maxLines	Use to set the maximum lines Textview can have.
24	android:minLines	Use to set the minimum lines Textview can have.
25	android:maxWidth	Use to set the maximum width Textview can have.
26	android:minWidth	Use to set the minimum width Textview can have
27	android:text	Use to set the text of the Textview

28	android:textAllCaps	Use this attribute when you want to show text in capital letters. It takes true or false value only. Default is false.
29	android:textColor	Use to set the color of the text.
30	android:textSize	Use to set the size of the text.
31	android:textStyle	Use to set the style of the text. For example, bold, italic, bolditalic.
32	android:typeface	Use to set typeface of the text. For example, normal, sans, serif, monospace.
33	android:width	Use to set width of the TextView.

2. EditText :

An Android EditText widget is user interface that are used to take input from user and modify the text. While defining EditText widget, we can also specify **android:inputType** attribute. Based on the value provided in this attribute, keyboard type shown also changes acceptable characters and appearance of the edit text.

Different attributes of Android EditText Widget

Different attributes that are used to customise the EditText are listed below. However, you can check android official documentation for EditText to see the complete list of it's attributes. Here, we are listing the commonly used attributes.

Attributes in EditText are inherited from TextView and View. Some of the popular attributes are –

Sr.	XML Attributes	Description
1	android:background	Sets background to this View.
2	android:backgroundTint	Sets tint to the background.
3	android:clickable	Set true when you want to make this View clickable. Otherwise, set false.
4	android:drawableBottom	Sets drawable to bottom of the text.

5	android:drawableEnd	Sets drawable to end of the text.
6	android:drawableLeft	Sets drawable to left of the text.
7	android:drawablePadding	Sets padding to drawable.
8	android:drawableRight	Sets drawable to right of the text.
9	android:drawableStart	Sets drawable to start of the text.
10	android:drawableTop	Sets drawable to top of the text.
11	android:elevation	Sets elevation to this view.
12	android:gravity	Sets gravity of the text. For example, center, horizontal_center, vertical_center etc.
13	android:height	Sets height of the EditText.
14	android:hint	Hint to be shown when there is no text in the EditText.
15	android:inputMethod	It sets, it specifies that edittext should use specified input method.
16	android:inputType	This is used to define what are the types of data that can be entered by the user for this View. For example, Phone, Password, Number, Date, Time etc. Characters acceptable through keyboard will also change accordingly.
17	android:lines	If you want to set height of the View by number of lines, you can do it using this attribute. For example, android:lines=”2”, it means height of View will be 2 lines.
18	android:maxHeight	Sets maximum height of the View.
19	android:minHeight	Sets minimum height of the View.
20	android:maxLength	Sets maximum character length that can be entered in the View.
21	android:maxLines	Sets maximum lines this View can have.
22	android:minLines	Sets minimum lines this View can have.

23	android:maxWidth	Sets maximum width this View can have.
24	android:minWidth	Sets minimum width this View can have.
25	android:numeric	If sets, it specifies that EditText has numeric input method.
26	android:password	Use this attribute if you want to show the entered text as password dots. For example, If you enter ABCD, it will be shown as ****.
27	android:phoneNumber	If set, specifies that this EditText has a phone number input method.
28	android:text	Sets the text of the EditText
29	android:textAllCaps	Use this attribute to show the text in capital letters.
30	android:textColor	Sets color of the text.
31	android:textSize	Sets size of the text.
32	android:textStyle	Sets style of the text. For example, bold, italic, bolditalic etc.
33	android:typeface	Sets typeface of the text. For example, normal, sans, serif, monospace.
34	android:width	Sets width of the TextView.

3. Button

Android Button is user interface that user can tap or click to perform some action.

Different Attributes Of Android Button Widget

Different attributes that are used in button are list below. However, if you want to see the complete list of attributes, you need to visit android official documentation on Button widget. Here, we are listing commonly used attributes.

Attributes in Button are inherited from TextView and View. Some of the popular attributes are –

Sr.	XML Attributes	Description
1	android:background	Sets background to this View.
2	android:backgroundTint	Sets tint to the background.
3	android:clickable	Sets true when you want to make this View clickable. Otherwise, set false.
4	android:drawableBottom	This is drawable to be drawn at bottom of the text.
5	android:drawableEnd	This is drawable to be drawn to end of the text.
6	android:drawableLeft	This is drawable to be drawn to left of the text.
7	android:drawablePadding	This is padding of the drawable.
8	android:drawableRight	This is drawable to be drawn to right of the text.
9	android:drawableStart	This is drawable to be drawn to start of the text.
10	android:drawableTop	This is drawable to be drawn at top of the text.
11	android:text	Sets the text of the EditText.
12	android:textAllCaps	Shows text in capital letters.
13	android:textColor	Sets color of the text.
14	android:textSize	Sets size of the text.
15	android:textStyle	Sets style of the text. For example, bold, italic, bolditalic etc.
16	android:typeface	Sets typeface of the text. For example, normal, sans, serif, monospace.

4. Toggle Button

Android Toggle button is a widget that are used to display checked/unchecked state as button with light indicator and by default accompanied with the text ON or OFF.

It is subclass of Compound button.

Different Attributes of Android Toggle Button Widget

Attributes in toggle button are –

Sr.	XML Attributes	Description
1	android:disabledAlpha	This is value of alpha you want to set when indicator is disabled.
2	android:textOff	Text to be shown when toggle button is in OFF state.
3	android:textOn	Text to be shown when toggle button is in ON state.

Attributes in Toggle Button are also inherited from TextView and Compound Button. Some of the popular attributes inherited from TextView are –

Sr.	XML Attributes	Description
1	android:background	Sets background to this View.
2	android:backgroundTint	Sets tint to the background.
3	android:clickable	Sets true when you want to make this View clickable. Otherwise, set false.
4	android:drawableBottom	This is drawable to be drawn at bottom of the text.
5	android:drawableEnd	This is drawable to be drawn to end of the text.
6	android:drawableLeft	This is drawable to be drawn to left of the text.
7	android:drawablePadding	This is padding of the drawable.
8	android:drawableRight	This is drawable to be drawn to right of the text.

9	android:drawableStart	This is drawable to be drawn to start of the text.
10	android:drawableTop	This is drawable to be drawn at top of the text.
11	android:text	Sets the text of the TextView.
12	android:textAllCaps	Shows text in capital letters.
13	android:textColor	Sets color of the text.
14	android:textSize	Sets size of the text.
15	android:textStyle	Sets style of the text. For example, bold, italic, bolditalic etc.
16	android:typeface	Sets typeface of the text. For example, normal, sans, monospace etc.

Attributes in Toggle Button inherited from Compound Button are –

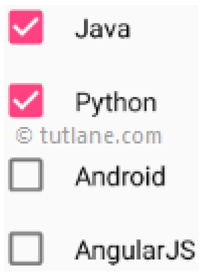
Sr.	XML Attributes	Description
1	android:button	Drawable used for the button graphic (for example, checkbox and radio button).
2	android:buttonTint	Tint to apply to the button graphic.

5. Checkbox:

In android, CheckBox is a two-states button that can be either checked (ON) or unchecked (OFF) and it will allow users to toggle between the two states (ON / OFF) based on the requirements.

Generally, we can use multiple CheckBox controls in android application to allow users to select one or more options from the set of values.

Following is the pictorial representation of using CheckBox control in android applications.



By default, the android CheckBox will be in the OFF (Unchecked) state. We can change the default state of CheckBox by using android:checked attribute.

In case, if we want to change the state of CheckBox to ON (Checked), then we need to set android:checked = “true” in our XML layout file.

In android, we can create CheckBox control in two ways either in the XML layout file or create it in the Activity file programmatically.

Create CheckBox in XML Layout File

Following is the sample way to define CheckBox control in XML layout file in android application.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent" android:layout_height="match_parent">
<CheckBox
   android:id="@+id/chk1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:text="Java" /> </RelativeLayout>
```

If you observe above code snippet, here we defined CheckBox control and setting CheckBox state ON using android:checked attribute in xml layout file.

Handle Android CheckBox Click Events

Generally, whenever the user clicks on CheckBox to Select or Deselect the CheckBox object will receive an on-click event.

Define CheckBox Click Event in Activity File

In android, we can define CheckBox click event programmatically in Activity file rather than XML layout file.

To define checkbox click event programmatically, create View.OnClickListener object and assign it to the button by calling setOnClickListener(View.OnClickListener) like as shown below.

```
CheckBox chk = (CheckBox) findViewById(R.id.chk1);
chk.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        boolean checked = ((CheckBox) v).isChecked();
        // Check which checkbox was clicked
        if (checked){
            // Do your coding
        }
        else{
            // Do your coding
        }
    }
});
```

This is how we can handle CheckBox click events in android applications based on our requirements.

Android CheckBox Control Attributes

The following are some of the commonly used attributes related to CheckBox control in android applications.

Attribute	Description
android:id	It is used to uniquely identify the control
android:checked	It is used to specify the current state of checkbox
android:gravity	It is used to specify how to align the text like left, right, center, top, etc.
android:text	It is used to set the text for a checkbox.
android:textColor	It is used to change the color of text.
android:textSize	It is used to specify the size of text.

Attribute	Description
android:textStyle	It is used to change the style (bold, italic, bolditalic) of text.
android:background	It is used to set the background color for checkbox control.
android:padding	It is used to set the padding from left, right, top and bottom.
android:onClick	It's the name of the method to invoke when the checkbox clicked.
android:visibility	It is used to control the visibility of control.

Android CheckBox Control Example
Following is the example of defining multiple CheckBox controls and one Button control in LinearLayout to get the selected values of CheckBox controls when we click on Button in the android application.

Create a new android application using android studio and give names as CheckBoxExample.

Now open an activity_main.xml file from \res\layout path and write the code like as shown below

activity_main.xml
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <CheckBox
    android:id="@+id/chkJava"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:layout_marginTop="150dp"
    android:layout_marginLeft="100dp"
    android:text="Java"
    android:onClick="onCheckboxClicked"/>
  <CheckBox
    android:id="@+id/chkPython"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_marginLeft="100dp"
        android:text="Python"
        android:onClick="onCheckboxClicked"/>
    <CheckBox
        android:id="@+id/chkAndroid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_marginLeft="100dp"
        android:text="Android"
        android:onClick="onCheckboxClicked"/>
    <CheckBox
        android:id="@+id/chkAngular"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_marginLeft="100dp"
        android:text="AngularJS"
        android:onClick="onCheckboxClicked"/>
    <Button
        android:id="@+id/getBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:text="Get Details" />
</LinearLayout>
```

If you observe above code we created a multiple CheckBox controls and one Button control in XML Layout file.

Once we are done with the creation of layout with required controls, we need to load the XML layout resource from our activity onCreate() callback method, for that open main activity file MainActivity.java from \java\com.tutlane.checkboxexample path and write the code like as shown below.

MainActivity.java

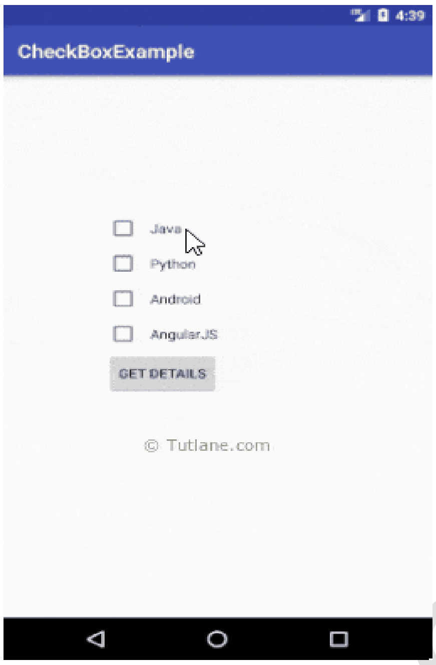
```
package com.tutlane.checkboxexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    CheckBox android, java, angular, python;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        android = (CheckBox)findViewById(R.id.chkAndroid);
        angular = (CheckBox)findViewById(R.id.chkAngular);
        java = (CheckBox)findViewById(R.id.chkJava);
        python = (CheckBox)findViewById(R.id.chkPython);
        Button btn = (Button)findViewById(R.id.getBtn);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String result = "Selected Courses";
                if(android.isChecked()){
                result += "\nAndroid";
                }
                if(angular.isChecked()){
                    result += "\nAngularJS";
                }
                if(java.isChecked()){
                    result += "\nJava";
                }
                if(python.isChecked()){
                    result += "\nPython";
                }
                Toast.makeText(getApplicationContext(), result, Toast.LENGTH_SHORT).show();
            }
        });
    }
    public void onCheckboxClicked(View view) {
        boolean checked = ((CheckBox) view).isChecked();
        String str="";
        // Check which checkbox was clicked
        switch(view.getId()) {
            case R.id.chkAndroid:
                str = checked?"Android Selected":"Android Deselected";
                break;
            case R.id.chkAngular:
                str = checked?"AngularJS Selected":"AngularJS Deselected";
                break;
            case R.id.chkJava:
                str = checked?"Java Selected":"Java Deselected";
                break;
            case R.id.chkPython:
                str = checked?"Python Selected":"Python Deselected";
                break;
        }
        Toast.makeText(getApplicationContext(), str, Toast.LENGTH_SHORT).show();
    }
}
```

If you observe above code we are calling our layout using setContentView method in the form of R.layout.layout_file_name in our activity file. Here our xml file name is activity_main.xml so we used file name activity_main and we are getting the status of CheckBox controls when they Select / Deselect and getting the selected CheckBox control values on Button click.

Generally, during the launch of our activity, onCreate() callback method will be called by android framework to get the required layout for an activity.

Output of Android CheckBox Example

When we execute the above example using the android virtual device (AVD) we will get a result like as shown below.



If you observe above result, we are able to get the status of checkboxes while selecting / deselecting and getting all the selected CheckBox values on button click.

This is how we can use CheckBox control in android applications to allow users to select one or more options based on our requirements.

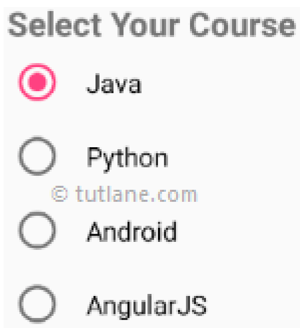
6. **RadioButton:**

In android, **Radio Button** is a two-states button that can be either checked or unchecked and it's the same as CheckBox control, except that it will allow only one option to select from the group of options.

The user can press or click on the radio button to make it select. In android, CheckBox control allow users to change the state of control either Checked or Unchecked but the radio button cannot be unchecked once it is checked.

Generally, we can use **RadioButton** controls in an android application to allow users to select only one option from the set of values.

Following is the pictorial representation of using **RadioButton** control in android applications.



In android, we use radio buttons with in a **RadioGroup** to combine multiple radio buttons into one group and it will make sure that users can select only one option from the group of multiple options.

By default, the android **RadioButton** will be in **OFF** (**Unchecked**) state. We can change the default state of **RadioButton** by using **android:checked** attribute.

In case, if we want to change the state of **RadioButton** to **ON** (**Checked**), then we need to set **android:checked = "true"** in our XML layout file.

In android, we can create **RadioButton** control in two ways either in the XML layout file or create it in the Activity file programmatically.

Create RadioButton in XML Layout File:
Following is the sample way to define **RadioButton** control using **RadioGroup** in the XML layout file in the android application.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent" android:layout_height="match_parent">
<RadioGroup
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical">
   <RadioButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Java"
      android:checked="true"/>
</RelativeLayout>
```

If you observe above code snippet, here we defined **RadioButton** control and setting RadioButton state **ON** using **android:checked** attribute in xml layout file.

Handle Android RadioButton Click Events:
Generally, whenever the user click on **RadioButton** to Select or Deselect the **RadioButton** object will receives an **on-click** event.

In android, we can define RadioButton click event in two ways either in the XML layout file or create it in Activity file programmatically.

Define RadioButton Click Event in Activity File
In android, we can define **RadioButton** click event programmatically in Activity file rather than XML layout file.

To define **RadioButton** click event programmatically, create **View.OnClickListener** object and assign it to the button by calling **setOnClickListener(View.OnClickListener)** like as shown below.

```
RadioButton rdb = (RadioButton) findViewById(R.id.radiobutton1);
rdb.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        boolean checked = ((RadioButton) v).isChecked();
        // Check which radiobutton was pressed
        if (checked){
            // Do your coding
        }
        else{
            // Do your coding
        }
    }
});
```

This is how we can handle **RadioButton** click events in android applications based on our requirements.

Android RadioButton Control Attributes:
Following are the some of commonly used attributes related to **RadioButton** control in android applications.

Attribute	Description
android:id	It is used to uniquely identify the control
android:checked	It is used to specify the current state of radio button
android:gravity	It is used to specify how to align the text like left, right, center, top, etc.

Attribute	Description
android:text	It is used to set the text for the radio button.
android:textColor	It is used to change the color of text.
android:textSize	It is used to specify the size of the text.
android:textStyle	It is used to change the style (bold, italic, bolditalic) of text.
android:background	It is used to set the background color for radio button control.
android:padding	It is used to set the padding from left, right, top and bottom.
android:onClick	It's the name of the method to invoke when the radio button clicked.
android:visibility	It is used to control the visibility of control.

Android RadioButton Control Example
Following is the example of defining a multiple **RadioButton** controls, one TextView control and one Button control in RelativeLayout to get the selected values of RadioButton controls when we click on Button in the android application.

Create a new android application using android studio and give names as **RadioButtonExample**.

Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent" android:layout_height="match_parent">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="150dp"
      android:layout_marginLeft="100dp"
      android:textSize="18dp"
      android:text="Select Your Course"
      android:textStyle="bold"
      android:id="@+id/txtView"/>
<RadioGroup
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:id="@+id/rdGroup"
    android:layout_below="@+id/txtView">
    <RadioButton
        android:id="@+id/rdbJava"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_marginLeft="100dp"
        android:text="Java"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton
        android:id="@+id/rdbPython"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_marginLeft="100dp"
        android:text="Python"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton
        android:id="@+id/rdbAndroid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_marginLeft="100dp"
        android:text="Android"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton
        android:id="@+id/rdbAngular"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:layout_marginLeft="100dp"
        android:text="AngularJS"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
    <Button
        android:id="@+id/getBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:layout_below="@+id/rdGroup"
        android:text="Get Course" />
</RelativeLayout>
```

If you observe above code we created a multiple **RadioButton** controls, one TextView control and one Button control in XML Layout file.

Once we are done with the creation of layout with required controls, we need to load the XML layout resource from our activity **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.radiobuttonexample** path and write the code like as shown below.

MainActivity.java

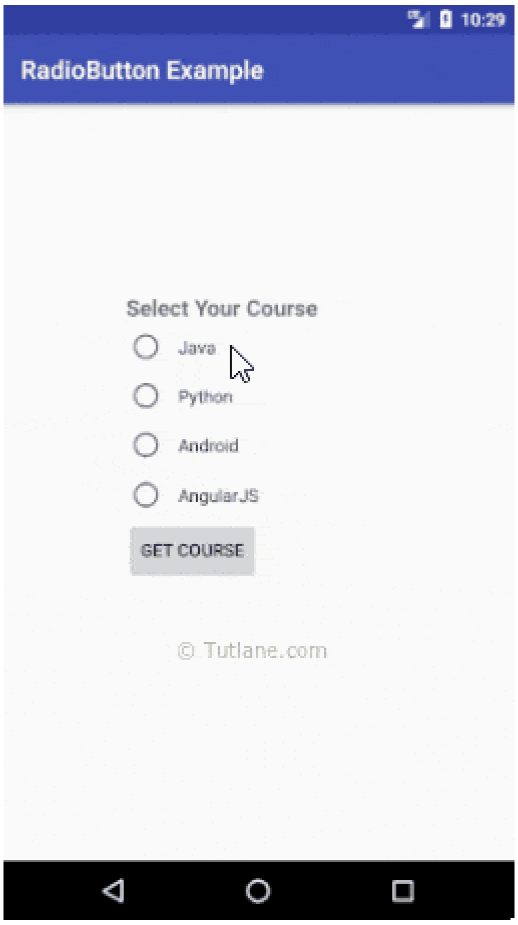
```
package com.tutlane.radiobuttonexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    RadioButton android, java, angular, python;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        android = (RadioButton)findViewById(R.id.rdbAndroid);
        angular = (RadioButton)findViewById(R.id.rdbAngular);
        java = (RadioButton)findViewById(R.id.rdbJava);
        python = (RadioButton)findViewById(R.id.rdbPython);
        Button btn = (Button)findViewById(R.id.getBtn);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String result = "Selected Course: ";
                result+= (android.isChecked())?"Android":(angular.isChecked())?"AngularJS":
(java.isChecked())?"Java":(python.isChecked())?"Python":"";
                Toast.makeText(getApplicationContext(), result, Toast.LENGTH_SHORT).sho
w();
            }
        });
    }
    public void onRadioButtonClicked(View view) {
        boolean checked = ((RadioButton) view).isChecked();
        String str="";
        // Check which radio button was clicked
        switch(view.getId()) {
            case R.id.rdbAndroid:
                if(checked)
                str = "Android Selected";
                break;
            case R.id.rdbAngular:
                if(checked)
                str = "AngularJS Selected";
                break;
            case R.id.rdbJava:
                if(checked)
                str = "Java Selected";
                break;
            case R.id.rdbPython:
                if(checked)
                str = "Python Selected";
                break;
        }
        Toast.makeText(getApplicationContext(), str, Toast.LENGTH_SHORT).show();
    }
}
```

If you observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout_file_name** in our activity file. Here our xml file name is **activity_main.xml** so we used file name **activity_main** and we are getting the status of **RadiButton** controls when they **Select** / **Deselect** and getting selected RadioButton control value on Button click.

Generally, during the launch of our activity, **onCreate()** callback method will be called by the android framework to get the required layout for an activity.

Output of Android RadioButton Example

When we run the above example using android virtual device (AVD) we will get a result like as shown below.



If you observe the above result, we are able to select only one option from the set of values and getting the selected **RadioButton** value on button click.

This is how we can use **RadioButton** control in android applications to allow users to select only one option from a set of values based on our requirements.

7. Spinner Control (Dropdown List):

In android, **Spinner** is a view that allows a user to select one value from the list of values. The spinner in android will behave same as a dropdown list in other programming languages.

Generally, the android spinners will provide a quick way to select one item from the list of values and it will show a dropdown menu with a list of all values when we click or tap on it.

By default, the android spinner will show its currently selected value and by using **Adapter** we can bind the items to spinner objects.

Following is the pictorial representation of using **spinner** in android applications.

We can populate our **Spinner** control with list of choices by defining an **ArrayAdapter** in our Activity file.

Generally, the **Adapter** pulls data from sources such as an array or database and converts each item into a result view and that's placed into the list.

Android Adapter

In android, **Adapter** will act as an intermediate between the data sources and adapter views such as **ListView**, **Gridview** to fill the data into adapter views. The adapter will hold the data and iterates through an items in data set and generate the views for each item in the list.

Generally, in android we have a different types of adapters available to fetch the data from different data sources to fill the data into adapter views, those are

Adapter	Description
ArrayAdapter	It will expect an Array or List as input.
CurosrAdapter	It will accepts an instance of a cursor as an input.
SimpleAdapter	It will accept a static data defined in the resources.
BaseAdapter	It is a generic implementation for all three adapter types and it can be used for ListView, Gridview or Spinners based on our requirements

Now we will see how to create spinner or dropdownlist in android applications.

Create Android Spinner in XML Layout File

In android, we can create **Spinner** in XML layout file using **<Spinner>** element with different attributes like as shown below.

```
<Spinner android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

Populate Android Spinner with Values

To populate spinner with list of values, we need to specify spinner adapter, such as an **ArrayAdapter** in activity file like as shown below.

```
String[] users = { "Suresh Dasari", "Trishika Dasari", "Rohini Alavala", "Praveen
Kumar", "Madhav Sai" };
Spinner spin = (Spinner) findViewById(R.id.spinner1);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.sim
ple_spinner_item, users);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spin.setAdapter(adapter);
```

This is how we can define and bind data to Spinner control in android applications. Now we will see complete example of using spinner control android applications.

Android Spinner Example

Following is the example of defining a one **Spinner** control, one TextView control in RelativeLayout to show the list of user details in android application.

Create a new android application using android studio and give names as **SpinnerExample**. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <TextView
        android:id="@+id/txtVw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dp"
        android:layout_marginTop="150dp"
        android:text="Select User:"
        android:textStyle="bold"
        android:textSize="15dp" />
    <Spinner
        android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/txtVw"
        android:layout_toRightOf="@+id/txtVw" />
</RelativeLayout>
```

If you observe above code we created a one **Spinner** control and one TextView control in XML Layout file.

Once we are done with the creation of layout with required controls, we need to load the XML layout resource from our activity **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.spinnerexample** path and write the code like as shown below.

MainActivity.java

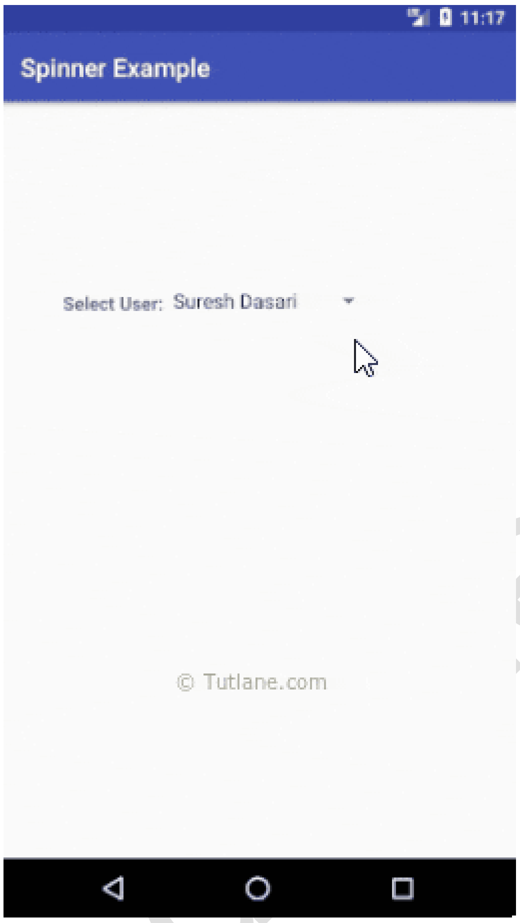
```
package com.tutlane.spinnerexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements AdapterView.OnItemS
electedListener {
String[] users = { "Suresh Dasari", "Trishika Dasari", "Rohini Alavala", "Praveen
Kumar", "Madhav Sai" };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner spin = (Spinner) findViewById(R.id.spinner1);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.la
yout.simple_spinner_item, users);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_i
tem);
        spin.setAdapter(adapter);
        spin.setOnItemSelectedListener(this);
    }
    @Override
    public void onItemSelected(AdapterView<?> arg0, View arg1, int position,long i
d) {
        Toast.makeText(getApplicationContext(), "Selected User: "+users[position]
,Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO - Custom Code
    }
}
```

If you observe the above code we are calling our layout using the **setContentView** method in the form of **R.layout.layout_file_name** in our activity file. Here our XML file name is **activity_main.xml** so we used file name **activity_main** and binding the list of values to **Spinner** control using **ArrayAdapter**.

Generally, during the launch of our activity, the **onCreate()** callback method will be called by the android framework to get the required layout for an activity.

Output of Android Spinner Example

When we run the above example using an android virtual device (AVD) we will get a result like as shown below.



If you observe above result, our spinner control is like dropdown list in other programming languages and we are able to get the selected user details in android application.

This is how we can use **Spinner** control in android applications to allow users to select one value from the list of values based on our requirements.

8. What is default and custom checkbox control in Android Studio :

Default Checkbox:

Default Checkbox is already explained in question no:6. It is nothing but the default state and design of the checkbox which is pre-defined in Android Studio. See explanation in question no:6.

Custom Checkbox:

When we change or do any modification in default checkbox control in android studio i.e. modifying design or other properties of it then this checkbox called as custom checkbox.
Android provides facility to customize the UI of view elements rather than default. You are able to create custom CheckBox in android. So, you can add some different images of checkbox on the layout.

Example of Custom CheckBox

In this example, we create both default as well as custom checkbox. Add the following code in activity_main.xml file.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.customcheckbox.MainActivity">


    <TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:textSize="25dp"
        android:text="Default Check Box"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
```

```
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New CheckBox"
    android:id="@+id/checkBox"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="46dp" />

<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New CheckBox"
    android:id="@+id/checkBox2"
    android:layout_below="@+id/checkBox"
    android:layout_alignLeft="@+id/checkBox"
    android:layout_alignStart="@+id/checkBox" />

<View
    android:layout_width="fill_parent"
    android:layout_height="1dp"
    android:layout_marginTop="200dp"
    android:background="#B8B894"
    android:id="@+id/viewStub" />

<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="CheckBox 1"
    android:id="@+id/checkBox3"
    android:button="@drawable/customcheckbox"
    android:layout_below="@+id/viewStub"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="58dp" />

<CheckBox
    android:layout_width="wrap_content"
```

```xml
        android:layout_height="wrap_content"
        android:text="CheckBox 2"
        android:id="@+id/checkBox4"
        android:button="@drawable/customcheckbox"
        android:layout_below="@+id/checkBox3"
        android:layout_alignLeft="@+id/checkBox3"
        android:layout_alignStart="@+id/checkBox3" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textSize="25dp"
        android:text="Custom Check Box"
        android:id="@+id/textView"
        android:layout_alignTop="@+id/viewStub"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show Checked"
        android:id="@+id/button"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Now implement a selector in another file (checkbox.xml) under drawable folder which customizes the checkbox.

checkbox.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:drawable="@drawable/checked" />
    <item android:state_checked="false" android:drawable="@drawable/unchecked"/>
</selector>
```

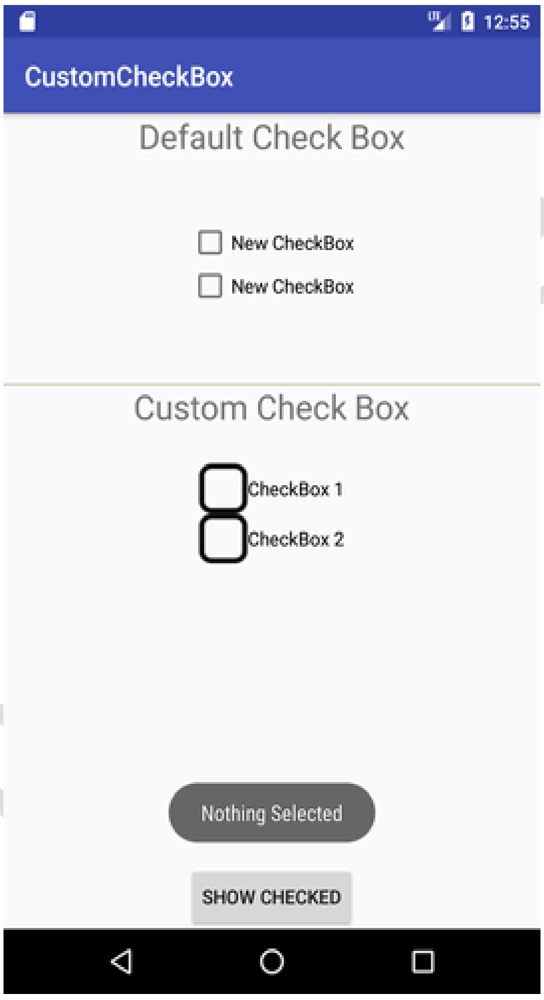
Activity class

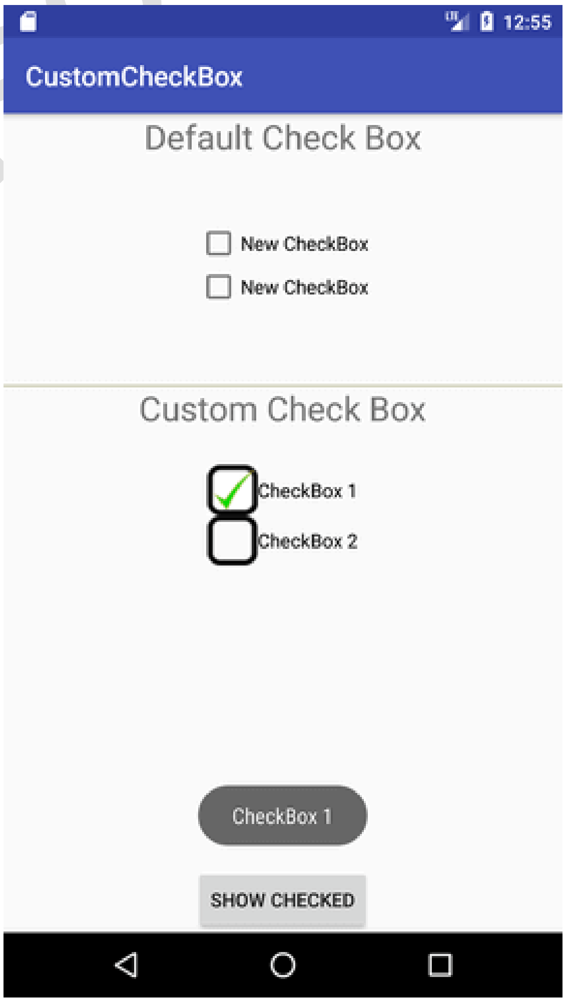
File: MainActivity.java

```
package example.javatpoint.com.customcheckbox;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    CheckBox cb1,cb2;
    Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        cb1=(CheckBox)findViewById(R.id.checkBox3);
        cb2=(CheckBox)findViewById(R.id.checkBox4);
        button=(Button)findViewById(R.id.button);

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                StringBuilder sb=new StringBuilder("");

                if(cb1.isChecked()){
                    String s1=cb1.getText().toString();
                    sb.append(s1);
                }

                if(cb2.isChecked()){
                    String s2=cb2.getText().toString();
                    sb.append("\n"+s2);

               }
```

```
                if(sb!=null && !sb.toString().equals("")){
                    Toast.makeText(getApplicationContext(), sb, Toast.LENGTH_LONG).show();

                }
                else{
                    Toast.makeText(getApplicationContext(),"Nothing Selected", Toast.LENGTH_LONG).
show();
                }

            }

        });
    }
}
```

Output





9. What is custom RadioButton and dynamic RadioButton control in Android Studio:

Custom RadioButton:

When we change or do any modification in default checkbox control in android studio i.e. modifying design or other properties of it then this checkbox called as custom checkbox.

Android provides facility to customize the UI of view elements rather than default. You are able to create custom CheckBox in android. So, you can add some different images of checkbox on the layout.

Example of Custom RadioButton

Let's see an example of custom RadioButton.

activity_main.xml

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context="com.example.test.customradiobutton.MainActivity">

    <TextView
        android:id="@+id/tv"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:gravity="center_horizontal"
        android:textSize="25dp"
        android:text="Customized Radio Buttons" />
    <!--   Customized RadioButtons  -->

<RadioGroup
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/radioGroup">

<RadioButton
          android:id="@+id/radioMale"
         android:layout_width="fill_parent"
          android:layout_height="wrap_content"
         android:text="  Male"
          android:layout_marginTop="10dp"
         android:checked="false"
          android:button="@drawable/custom_radio_button"
         android:textSize="20dp" />

        <RadioButton
          android:id="@+id/radioFemale"
         android:layout_width="fill_parent"
          android:layout_height="wrap_content"
         android:text="   Female"
          android:layout_marginTop="20dp"
         android:checked="false"
          android:button="@drawable/custom_radio_button"
         android:textSize="20dp" />
    </RadioGroup>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show Selected"
        android:id="@+id/button"
        android:onClick="onclickbuttonMethod"
        android:layout_gravity="center_horizontal" />

    </LinearLayout>
```

File: checkbox.xml

```
<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

  <item android:state_checked="true" android:drawable="@drawable/checkedradiobutton" />

   <item android:state_checked="false" android:drawable="@drawable/unchekedradiobutton" />

</selector>
```

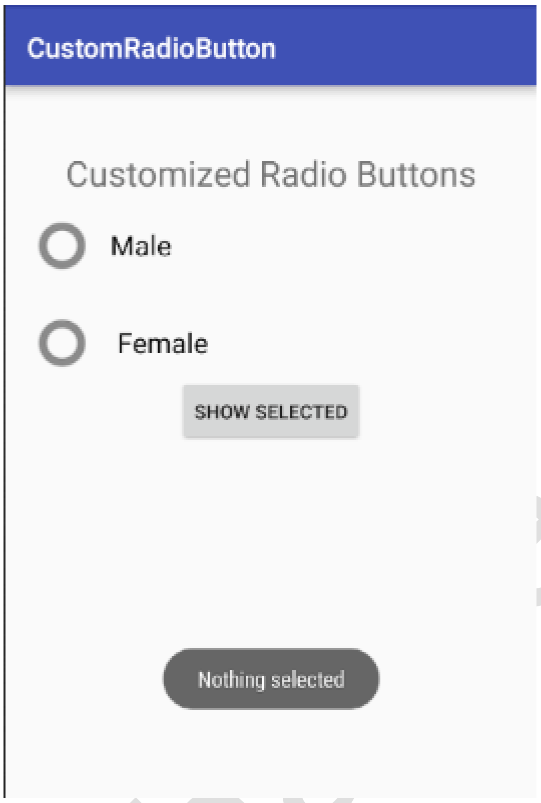
Activity class

File: MainActivity.java

```
package com.example.test.customradiobutton;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   Button button;
   RadioButton genderradioButton;
   RadioGroup radioGroup;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
   }

  public void onclickbuttonMethod(View v){
      int selectedId = radioGroup.getCheckedRadioButtonId();
      genderradioButton = (RadioButton) findViewById(selectedId);
      if(selectedId==-1){
         Toast.makeText(MainActivity.this,"Nothing selected", Toast.LENGTH_SHORT).show();
```

```
        }
        else{
            Toast.makeText(MainActivity.this,genderradioButton.getText(), Toast.LENGTH_SHORT).s
how();
        }

    }
}
```

Output


CustomRadioButton

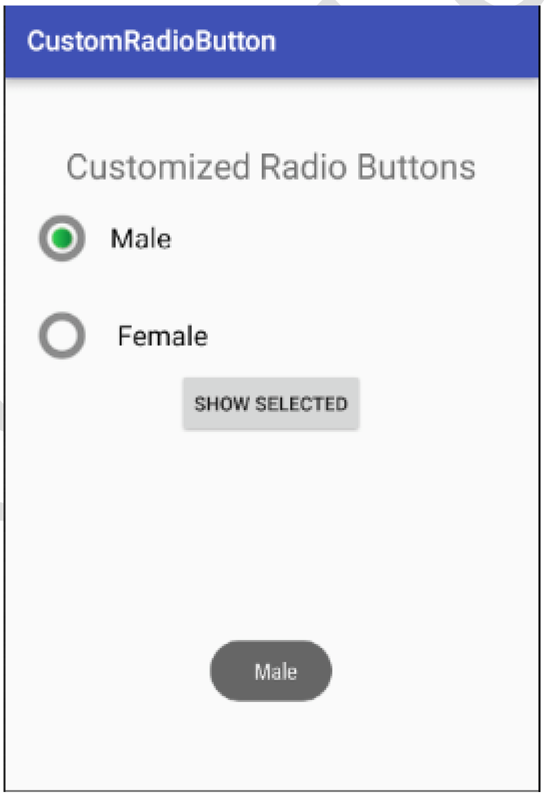
Customized Radio Buttons

Male

Female

SHOW SELECTED

Nothing selected


CustomRadioButton

Customized Radio Buttons

Male

Female

SHOW SELECTED

Male

10. Explain AlertDialog control in details:

Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue.

Android AlertDialog is composed of three regions: title, content area and action buttons.

Android AlertDialog is the subclass of Dialog class.

Methods of AlertDialog class

Method	Description
public AlertDialog.Builder setTitle(CharSequence)	This method is used to set the title of AlertDialog.
public AlertDialog.Builder setMessage(CharSequence)	This method is used to set the message for AlertDialog.
public AlertDialog.Builder setIcon(int)	This method is used to set the icon over AlertDialog.

Android AlertDialog Example

Let's see a simple example of android alert dialog.

activity_main.xml

You can have multiple components, here we are having only a textview.

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.alertdialog.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        android:text="Close app"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

strings.xml

Optionally, you can store the dialog message and title in the strings.xml file.

File: strings.xml

```
<resources>
<string name="app_name">AlertDialog</string>
<string name="dialog_message">Welcome to Alert Dialog</string>
<string name="dialog_title">Alert Dialog</string>
</resources>
```

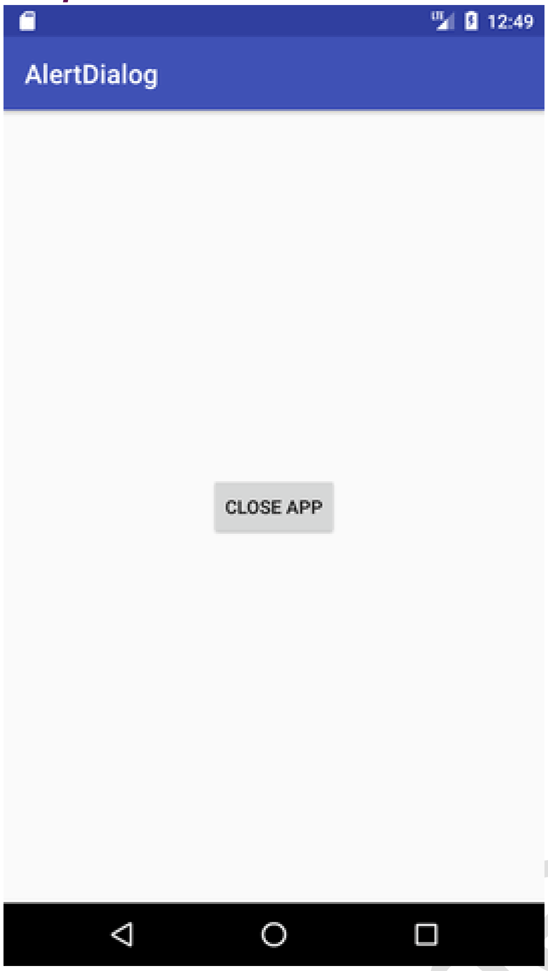
Activity class

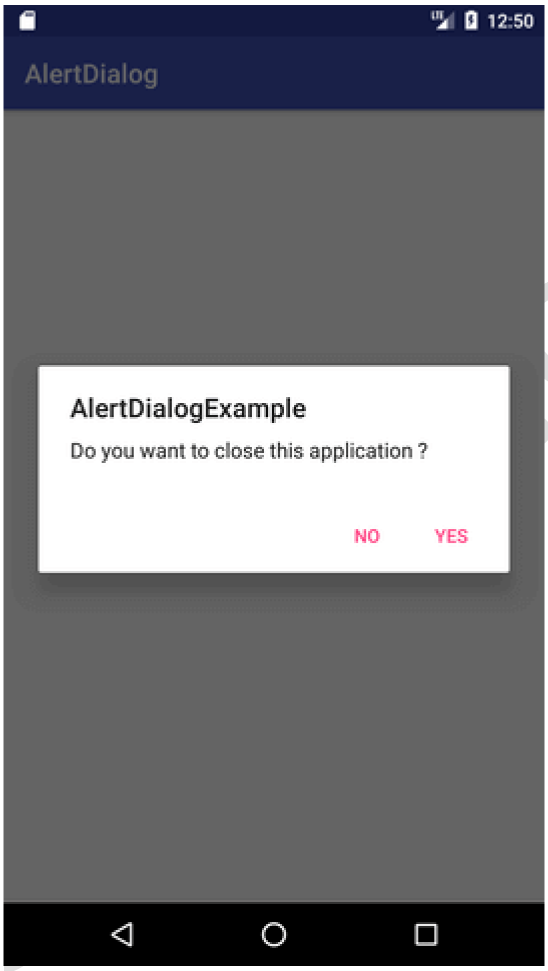
Let's write the code to create and show the AlertDialog.

File: MainActivity.java

```
package example.javatpoint.com.alertdialog;

import android.content.DialogInterface;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.app.AlertDialog;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    Button closeButton;
    AlertDialog.Builder builder;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        closeButton = (Button) findViewById(R.id.button);
        builder = new AlertDialog.Builder(this);
        closeButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                //Uncomment the below code to Set the message and title from the strings.xml file
                builder.setMessage(R.string.dialog_message) .setTitle(R.string.dialog_title);

                //Setting message manually and performing action on button click
                builder.setMessage("Do you want to close this application ?")
                        .setCancelable(false)
                        .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                finish();
                                Toast.makeText(getApplicationContext(),"you choose yes action for alertbox",
                                Toast.LENGTH_SHORT).show();
                            }
                        })
                        .setNegativeButton("No", new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {
                                //  Action for 'NO' Button
                                dialog.cancel();
                                Toast.makeText(getApplicationContext(),"you choose no action for alertbox",
                                Toast.LENGTH_SHORT).show();
                            }
                        });
                //Creating dialog box
                AlertDialog alert = builder.create();
                //Setting the title manually
                alert.setTitle("AlertDialogExample");
                alert.show();
            }
        });
    }
}
```

Output:





AlertDialogExample

Do you want to close this application ?